CMPT 135

Sample Midterm Exam 2 **Solutions**

Based on Spring 2022 midterm

This is a **50 minute closed book exam**: notes, books, computers, calculators, electronic devices, etc. are **not** permitted. Do not speak to any other students during their exam or look at their work. Please remain seated and **raise your hand** if you have a question.

Basic C++

(10 marks) Call a character an **SFU character** if it is one of the 3 lowercase letters *s*, *f*, or *u*.

Write a function called `most_common(x)` that returns the most frequently occurring SFU character in the `string x`. If two or more letters tie for most commonly occurring, return the one that comes first *alphabetically*, i.e. first *f*, then *s*, then *u*.

For example, `most_common("vacuums!")` returns `'u'`, and `most_common("ufsufs")` returns `'f'`.

For this question:

- Use only basic C++ in your solution as discussed in the course.
- **Do not** use any #include-ed code except for `#include <string>`.
- You can use the function `max(a, b)`, which returns the maximum of two integers `a` and `b`. For example, `max(9, 4)` returns 9.

Sample Solution:

```
char most_common(const string& x) {

    int s = 0;
    int f = 0;
    int u = 0;

    for(char c : x) {

        if (c == 's') s++;
        else if (c == 'f') f++;
        else if (c == 'u') u++;
    }

    int most = max(s, max(f, u));

    if (f == most) {
        return 'f';
    } else if (s == most) {
        return 's';
    } else {
        return 'u';
    }
}
```

Pointers and Dynamic Memory

For the following questions, assume `#include <vector>` and `using namespace std;` have already been written. You **don't** need to write the code fragments inside a function.

a) (3 marks) Write a fragment of code that **defines** a `vector` of `int` pointers called `v` and initializes it to contain 100 `int` pointer values. Make each `int` pointer point to a newly created `int` on the free store. The `int`s should be in order from 0 to 99.

```
vector<int*> v(100);
for(int i = 0; i < v.size(); i++) {
    v[i] = new int(i);
}
```

b) (3 marks) Suppose `v` is a `vector` of 0 or more `int` pointers, and none are null pointers. Write a fragment of code that uses a **while**-loop to print the `int`s that the pointers in `v` point to, one per line.

```
int i = 0;
while (i < v.size()) {
    cout << *v[i] << "\n";
    i++;
}
```

c) (4 marks) Suppose `v` is a `vector` of 0 or more `int` pointers, and **null pointers are allowed**. Write a fragment of code that uses a **for**-loop to calculate and print the **sum** of the `int`s that `v` points to. Treat null pointers as if they pointed to an `int` with value 1.

```
int sum = 0;
for(int i = 0; i < v.size(); i++) {
    if (v[i] == nullptr) {
        sum += 1;
    } else {
        sum += *v[i];
    }
}
cout << "sum = " << sum << "\n";
```

d) (2 marks) Suppose `v` is a `vector` of 0 or more `int` pointers, and every pointer points to a different `int` that was allocated on the free store with `new`. Write a fragment of code that de-allocates all the `int`s `v` points to so there are no memory leaks or other memory errors.

```
for(int i = 0; i < v.size(); i++) {
    delete v[i];
};
```

Classes

(20 marks) Write a class called `Candy` that stores the name (a `string`) and cost (a `double`) of a store-bought candy. Your class must have these features:

- All member variables are **private**.
- All methods are **public**.
- A **default constructor** that uses **member initialization** to set the candy name to "none", cost to -1, and prints the message "object created" to `cout`.
- A **constructor** that uses an **initialization list** to set the candy name and cost to values passed into the constructor. If the name is an empty string, or if the cost is less than 0, then it should throw an error using `cmpt::error`.
- A **copy constructor** that uses **constructor delegation** to set the candy name and cost to be the same as the name and cost of another passed-in `Candy` object.
- A **destructor** that prints the message "object deleted".
- **Getters** that return the cost of the candy, and the name of the candy.
- A **setter** that sets the cost of the candy to be a given `double`. If the given `double` is less than 0, then it should throw an error using `cmpt::error`.
- Define an `==` operator that tests if two `Candy` objects have the same name and cost. Importantly, define this `==` *outside* of the `Candy` class.
- Write this code neatly, and use good indentation and C++ style.

```
class Candy {
private:
    string name = "none";              // member initialization
    double cost = -1;

public:
    Candy() {                          // default constructor
        cout << "object created\n";
    }

    Candy(const string& n, double c)
    : name(n), cost(c)  // initializer list
    {
        if (name == "" || cost < 0) cmpt::error("bad data");
    }

    Candy(const Candy& other)          // copy constructor
    : Candy(other.name, other.cost)  // constructor delegation
    { }

    ~Candy() {                         // destructor
        cout << "object deleted\n";
    }

    // getters
    string get_name() const { return name; }
    double get_cost() const { return cost; }

    // setters
    void set_cost(double c) {
       if (c < 0) cmpt::error("cost can't be negative");
       cost = c;
     }
}; // class Candy


bool operator==(const Candy& a, const Candy& b) {
    return (a.get_name() == b.get_name())
        && (a.get_cost() == b.get_cost());
}
```